TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ANTTI KUKKO

KONVOLUUTIONEUROVERKKOJEN OPTIMOINTI

Kandidaatintyö

Tarkastaja: Pia Niemelä
6.1.2018

TIIVISTELMÄ

ANTTI KUKKO: Konvoluutioneuroverkkojen optimointi
Optimizing convolutional neural nets
Tampereen teknillinen yliopisto
Kandidaatintyö, 21 sivua
Tammikuu 2018
Tietotekniikan koulutusohjelma
Pääaine: Ohjelmistotekniikka
Tarkastaja: Pia Niemelä
Avainsanat: Neuroverkko, konvoluutioneuroverkko, optimointi, hahmontunnistus, syväoppiminen, kuvantunnistus

Tämä kandidaatintyö käsittelee konvoluutioneuroverkkojen optimointia. Työssä esitellään joitakin neuroverkkojen perusperiaatteista, ongelmatapauksia ja menetelmiä ongelmien ratkaisemiseksi.

SISÄLLYS

1. JOHDANTO

Viime aikoina koneoppiminen ja syväoppiminen ovat saaneet huomiota osakseen. Nämä tieteenalat ovat olleet olemassa jo pitkään, ja monet tällä hetkellä parrasvaloissa olevista menetelmistä ovat jo useita kymmeniä vuosia vanhoja. Moni alan perusteos on vuosituhannen edelliseltä puolelta, ja silloin tehdyt havainnot ovat edelleen aivan yhtä valideja nykyään. Tästä huolimatta vasta viimeaikaiset kehitysaskeleet koneoppimisessa ovat saaneet monet koneoppimisen menetelmät sellaiselle tasolle, että niitä voidaan käyttää reaalimaailman haasteiden ratkaisemiseen. Osaltaan tietokoneiden kasvanut suoritusteho on edesauttanut koneoppimismenetelmien käyttöönottoa käytännön sovelluksissa, sillä yhä monimutkaisempia laskutoimituksia on käytännöllistä ajaa jopa tavallisilla kuluttajatietokoneilla. Toisaalta viime vuosina on tehty erityisesti neuroverkkoihin liittyviä merkittäviä tieteellisiä läpimurtoja, jotka ovat nekin osittain mahdollistaneet viime vuosien edistyksen.

Viiden edellisen vuoden aikana yhä useampi ongelma on pystytty ratkaisemaan neuroverkkojen avulla. Esimerkiksi vielä kymmenen vuotta sitten erityisen hankalana pidetty ongelma, tietokoneen suorittama kuvantunnistus, on nykyään jo arkipäivää. Automaattinen kuvantunnistus on otettu käyttöön yhä useammissa eri käyttötarkoituksissa. Tästä on kiittäminen kehittyneitä kuvantunnistusmenetelmiä, joilla kuvantunnistuksen tarkkuus on saatu riittävän hyvälle tasolle. Myös esimerkiksi puheentunnistus ja tietyt ihmisten käytöstä oppivat algoritmit ovat kehittyneet paljon viime vuosien aikana.

Erityisesti syväoppiminen on nähty viime vuosien kehityksessä merkittävimpänä tekijänä. Aiemmin syväoppiminen koettiin hankalaksi ja epäkäytännölliseksi, mutta nykyinen tietämys kertoo toista tarinaa. Syväoppimiseen ja syvien neuroverkkojen opettamiseen liittyy silti omat ongelmansa. Tässä kandidaatintyössä käsitellään joitakin näistä ongelmista, ja esitetään tapoja minimoida ongelmien vaikutusta koneoppimissovelluksissa.

Työssä esitellään neuroverkkojen perusteoriaa ja erilaisia tapoja optimoida neuroverkkoja. Työn päätarkoitus on pyrkiä selvittämään miten neuroverkkoja pystytään kehittämään ja parantamaan olemassa olevilla optimointimenetelmillä. Teoriaosuu-

den tukena tullaan käyttämään alan kirjallisuutta. Työssä on kokeellinen osuus, jossa testataan optimointimenetelmiä käytännössä. Kokeellisen osuuden päätteeksi kokeen tuloksia arvioidaan ja pohditaan, minkä takia saatuihin tuloksiin päädyttiin ja millä asioilla siihen oli vaikutusta.

2. KONEOPPIMINEN YLEISESTI

Koneoppiminen on tietojenkäsittelytieteen alue, joka käsittelee tietokoneen kykyä oppia asioita ilman erillistä ohjelmointia. Koneoppiminen perustuu laajalti tietokoneen kykyyn huomata suurista määristä dataa tiettyjä malleja, ja näiden mallien avulla tietokone pystyy matkimaan ihmismäistä oppimista. Yksi tyypillinen esimerkki koneoppimisesta on kuvien tunnistaminen. Nykypäivän koneoppimisalgoritmit pystyvät esimerkiksi lukemaan käsin kirjoitettuja numeroita tai tunnistamaan liikennemerkkejä joko lähes yhtä hyvin, tai paremmin, kuin ihminen [1].

2.1 Opettaminen

Mitchellin mukaan tietokone oppii jonkin tietyn tehtävän T (task) tehokkuudella P (performance) kokemuksen E (experience) avulla, mikäli järjestelmä onnistuu luotettavasti parantamaan tehokkuuttaan P tehtävässä T kokemuksen E seurauksena [2]. Parannuksen on siis tapahduttava siitä syystä, että tietokone on altistettu jollekin kokemukselle. Kuvantunnistuksessa tehtävä voisi olla esimerkiksi liikennemerkkien tunnistaminen kuvista. Kokemus olisi liikennemerkkikuvien näyttäminen koneelle. Tehokkuusmittari voisi olla koneen oikein tunnistamien liikennemerkkien suhde väärin tunnistettuihin merkkeihin.

Koneoppimisalgoritmin opettamiseen kuuluu yleensä algoritmin optimointi opetusdataa vasten. Opetukseen käytetään siis kaikki tarjolla oleva opetusdata pyrkimyksenä saada algoritmi tuottamaan opetusdatan mukainen lopputulos. Mallin tehokkuutta mitataan määrittelemällä jokin tehokkuusmittari mallin toimivuudelle. Kirjallisuudessa esiintyviä tehokkuusmittareita ovat esimerkiksi keskiarvoistettu virheen neliösumma (mean square error) ja maksimitodennäköisyys (maximum likelihood). Näiden molempien metodien tarkoitus on mitata mallin ennustamaa ja todellista arvoa ja minimoida ero todellisen ja ennustetun arvon välillä. Tähän aiheeseen palataan luvuissa 3.3 ja 3.4.

2.2 Ylioppiminen

Ylioppiminen on yleinen ongelma koneoppimisessa. Ylioppimisella tarkoitetaan tilannetta, jossa jokin koneoppimisen malli on oppinut vain opetusdatan aineiston, eikä pysty yleistämään oppimaansa asiaa uusiin esimerkkeihin. Toisin sanoen malli pystyy tunnistamaan vain sellaisia esimerkkejä, jotka se on jo nähnyt. Kuvantunnistuksessa tämä tarkoittaisi, että malli pystyy tunnistamaan vain sellaiset kuvat, jotka mallille on jo näytetty, mutta jos sille näytetään uusi kuva, tunnistus tapahtuu enemmän tai vähemmän sattumanvaraisesti.

Ongelmana on, että jos mallin annetaan optimoida itseään opetusdataa vasten tarpeeksi perusteellisesti, se kehittää sellaisia yhteyksiä, joita ei oikeasti ole olemassa. Kuvantunnistuksessa tämä voisi tarkoittaa sitä, että malli alkaa yhdistää yksittäisten pikselien suhteita johonkin tiettyyn luokkaan, vaikka näin tarkasti määritelty yhteys ei pidä paikkaansa kuin yksittäisessä kuvassa kerrallaan. Algoritmi oppisi siis tunnistamaan juuri tietynlaisen kuvan tietyillä pikseliarvoilla oikeaan luokkaan, mutta kohdatessaan minkä tahansa muun kuvan, tämä tapa yhdistää kuva luokkaan epäonnistuisi täysin, koska yksittäiset pikselit eivät koskaan ratkaise sitä, mitä kuvassa esiintyy. [3, s. 110–116]

Ylioppimisongelman ratkaisemiseksi opetusdatan lisäksi käytetään testi- ja validointidataa. Opetusdataa käytetään mallin opettamiseen. Mallin parametrien säätäminen vaikuttaa tapaan, jolla malli oppii opetusdatan. Parametrien säätämisen apuna käytetään validointidataa. Opetusdatan opettamisen eri vaiheissa mallia testataan validointidatalla, ja tästä saatujen tuloksien perusteella parametreja säädetään johonkin suuntaan. Neuroverkkojen tapauksessa tämä tarkoittaisi verkon painojen säätämistä. Lopulta, kun malli on saatu opetettua ja tuloksiin ollaan tyytyväisiä, mallia aletaan testata testidatalla. Näin saadaan tulokset valmiille mallille ja voidaan arvioida, kuinka hyvin malli toimisi oikeassa maailmassa datalla, jota se ei ole koskaan nähnyt. Tässä vaiheessa on yleensä vielä mahdollista säätää mallin hyperparametreja testidatan antamien tuloksien perusteella. Hyperparametreja käsitellään kattavammin luvussa 4.5. [3, s. 110–116]

3. NEUROVERKKO

Neuroverkko on eräs koneoppimisen malli. Yksinkertaistettuna neuroverkko tarkoittaa suurta joukkoa funktioita, jotka on kytketty toisiinsa niin, että yhden funktion lopputulos vaikuttaa seuraaviin funktioihin. Funktioilla voi olla kertoimia, jolloin niiden antamat tulokset vaikuttavat eri tavalla tuloksiin. Neuroverkon opetuksessa kertoimille etsitään sopivia arvoja. Kertoimien painotuksien on tarkoitus mahdollistaa erilaisten piirteiden korostaminen ja toisenlaisten piirteiden merkityksen pienentäminen, kun yritetään ennustaa esimerkkidatasta oikeaa ratkaisua.

3.1 Historiaa

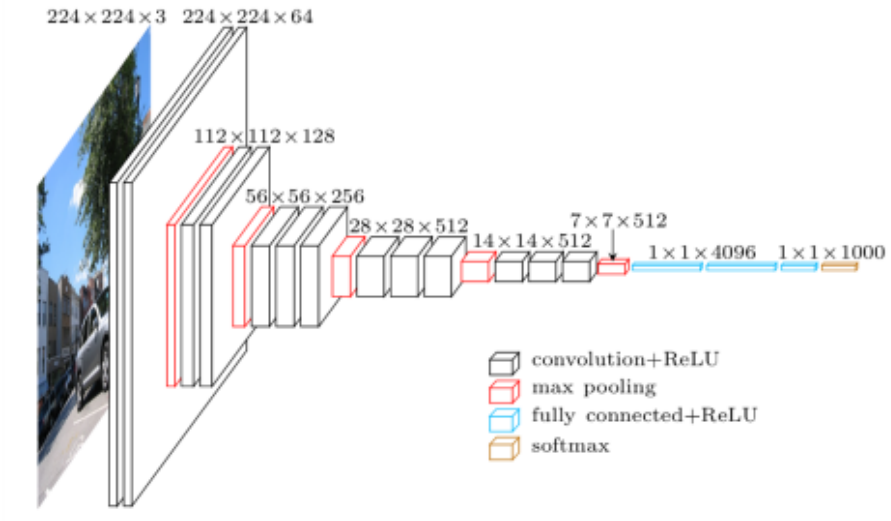
Yksi kuvantunnistuksen suurimmista haasteista on aina ollut piirteiden löytäminen. Piirteellä tarkoitetaan jotakin sellaista datan ominaisuutta tai säännöllisyyttä, joka on löydettävissä keskenään samankaltaisissa tapauksissa. Esimerkiksi ihmiskasvoja tunnistettaessa piirre voisi olla jokin geometrinen mittasuhde, josta ihmisen kasvonpiirteet koostuvat. Piirteitä voisivat olla esimerkiksi silmien symmetrinen asema, niiden suhde nenään tai kasvojen ulkoreunat. Piirteiden etsiminen oli pitkään manuaalista, ihmisten tekemää työtä. Kuvantunnistusta varten erilaisia piirteitä täytyi olla runsaasti, ja hyvien piirteiden löytäminen oli työlästä.

Toisaalta piirteet ovat kriittisessä roolissa kuvantunnistuksessa. Hyvien piirteiden avulla itse oppimismalli voidaan pitää yksinkertaisena, kun taas huonosti valitut piirteet saavat mallin oppimaan epäoptimaalisesti [4, s. 1]. Tärkeä syy neuroverkkojen suosiolle on, että neuroverkko löytää piirteet omatoimisesti ilman ihmisen apua. Tämä oli yksi niistä läpimurroista, joiden ansiosta kuvantunnistus on kehittynyt viimeisten vuosien aikana.

3.2 Neuroverkon rakenne

Neuroverkko koostuu kerroksista. Jokaisessa kerroksessa verkkoon syötettävälle datalle tehdään matemaattisia operaatioita, jotka muokkaavat dataa erilaiseen muotoon. Tämä erilainen muoto vastaa käytännössä piirteiden irroitusta, joka ennen

neuroverkkojen läpimurtoa oli tehtävä manuaalisesti. Kuvien tapauksessa tyypillinen konvoluutioneuroverkko pilkkoo kuvaa pienemmiksi harmaansävykuviksi, samalla kasvattaen kuvamatriisin värikanavadimensiota. RGB-kuvan kolme värikanavadimensiota kasvaa siis useaksi eri kerrokseksi. Tämän operaation jälkeen kyseessä ei ole enää RGB-kuva, vaan neuroverkon oppimista varten erityiseen muotoon muotoiltu kolmiulotteinen matriisi. Jos ihminen yrittäisi saada tästä matriisista selvää, se saattaisi vielä jossain määrin onnistua neuroverkon ensimmäisillä kerroksilla. Kerroksissa syvemmälle edetessä matriisin muoto muuttuu kuitenkin yhä abstraktimmaksi, ja lopulta ihminen ei pysty enää hahmottamaan yhtäläisyyksiä alkuperäisen kuvan kanssa. Kuvassa 3.1 on esitetty VGG16-neuroverkon rakenne.



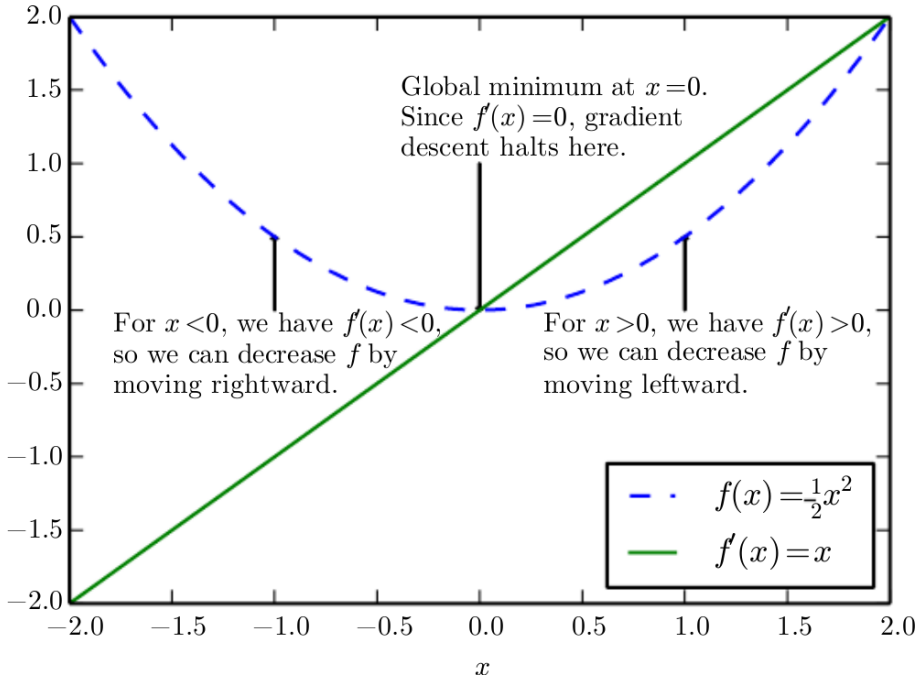
Kuva 3.1 *VGG16-neuroverkon rakenne. [5]*

Matriisin saatua lopullisen abstraktin muotonsa, siitä tehdään linkitys varsinaiseen vastaukseen. Kuvanluokitteluongelmaa ratkaistaessa kuva liitetään kuvaa vastaavan luokan tunnukseen. Tyypillisesti luokitteluongelmien tapauksessa oikea luokka esitetään yksiulotteisena matriisina, jossa on niin monta alkiota kuin opetusdatassa on luokkia. Jokainen matriisin alkio vastaa yhtä luokkaa, ja vastaukseksi annetaan todennäköisyysarvo kullekin luokalle. Syynä tälle järjestelylle on se, että virhefunktion määrittäminen helpottuu kun kyseessä on lineaarinen todennäköisyysasteikko binäärisen kyllä/ei-asteikon sijaan.

3.3 Gradienttien käyttö neuroverkoissa

Neuroverkkojen perusperiaatteen ymmärtämiseksi on hyvä ymmärtää gradientin toimintaperiaate ja sen rooli neuroverkoissa. Neuroverkkojen optimoinnissa on tyypillistä minimoida jotakin funktiota (hintafunktio, cost function). Tämän funktion gra-

dientin (derivaatan) avulla pystytään määrittämään mihin suuntaan funktion kuvaajaa täytyy liikkua, jotta päädytään edellistä pienempään arvoon. Siirtymällä infinitesimaalisen pieniä hyppyjä gradientin määrittämään suuntaan, löydetään lopulta lokaali minimikohta. Tämä tekniikka on nimeltään gradient descent.



Kuva 3.2 *Havainnollistava kuva gradient descent -menetelmästä. [3]*

Pisteitä, joissa kuvaaja on täysin vaakatasossa, kutsutaan kriittisiksi pisteiksi. Näissä pisteissä gradientin avulla ei saada tietoa uudesta suunnasta. Pistettä joka on kaikista arvoista alhaisin, kutsutaan globaaliksi minimiksi. Syväoppimisen kontekstissa ei ole aina mahdollista löytää globaalia minimiä. Verkon opettamisen kannalta voi joskus olla tehokkainta löytää tarpeeksi lähellä globaalia minimiä oleva lokaali minimikohta. [3, s. 82–84]

Gradientin laskeminen ei ole kovin hintava operaatio, mutta jos sama operaatio täytyisi suorittaa miljoonille opetusdatan eri tapauksille, kuluisi siihen jo huomattavasti aikaa. Sen sijaan, että gradient descent -tekniikkaa käytettäisiin jokaiselle opetusdatan alkiolle, sitä käytetään vaihe vaiheelta vain pienelle osalle saatavilla olevasta datasta. Perusteluna tälle on se, että toimiakseen algoritmin ei tarvitse tuottaa täydellisen tarkkaa tulosta. Riittää, kun gradientin suunta saadaan pääsääntöisesti määritettyä oikein. Niinpä on mahdollista säästää aikaa tekemällä operaatio vain datasta tasaisesti valituille alkioille, yleensä yhdestä muutamiin satoihin alkioihin kerrallaan [3, s. 151].

3.4 Hintafunktio

Gradientin avulla pystytään siis määrittämään suunta johon parametreja kannattaa liikuttaa. Seuraavaksi herää kuitenkin kysymys, mitä hintafunktio oikeastaan kuvaa? Hintafunktion tarkoitus on jollain tavalla määrittää kuinka hyvin haluttu tehtävä on opittu. Kuvanluokittelun tapauksessa haluttaisiin siis mitata, kuinka hyvin malli yhdistää kuvia oikeisiin luokkiin. Tämän suureen mittaamisessa käytetään suurimman todennäköisyyden funktiota (maximum likelihood estimation). Kyse on siis tavasta määrittää, miten lähelle mallin ennustamat todennäköisyydet osuvat oikeaa tulosta. Jos malli ennustaa oikean luokan todennäköisyydeksi pienen todennäköisyyden, malli on oppinut datan huonosti. Jos taas oikean luokan todennäköisyys on suuri, malli suoriutuu hyvin.

Negatiivinen logaritmitodennäköisyys (negative log likelihood) on eräs hyvin yleinen hintafunktio. Sen minimointi vastaa suurimman todennäköisyyden maksimointia [3, s. 178]. Pienen arvon saadessaan negatiivinen logaritmitodennäköisyys indikoi sitä, että arvioitu todennäköisyys oikealle luokalle on suuri. Vastaavasti suuria arvoja saadessaan negatiivinen logaritmitodennäköisyys indikoi mallin suoriutuneen huonosti tehtävästä. Syy funktion logaritmisuudelle liittyy olennaisesti katoavan gradientin ongelmaan. Logaritmi käyttäytyy lähellä nollaa eri tavalla kuin pelkkä todennäköisyysarvo käyttäytyisi. Tähän aiheeseen palataan luvussa 4.2.

Hintafunktion ohella käytetään usein jonkinlaista regularisointitermiä. Syynä tälle on se, että pelkkä hintafunktio ei pysty yleistämään testidatasta uuteen dataan [3, s. 218–219, 228]. Painoheikennys (weight decay) on yksi yleinen regularisointimenetelmä. Sen perusperiaate on lisätä hintafunktioon ylimääräinen regularisointitermi, joka aiheuttaa huonomman hintafunktion arvon suurilla painojen arvoilla. Weight decay -regularisointimenetelmän ansiosta painojen joukossa on vähemmän hyödyttömiä komponentteja, ja opetusdatan staattisen kohinan vaikutukset tuloksiin pienenevät [6]. Tämä regularisointimenetelmä on myös käytössä luvun 5 kokeessa.

Hintafunktiota voidaan minimoida luvussa 3.3 kuvatulla gradient descent -menetelmällä. Hintafunktiosta laskettuja gradientteja hyödynnetään puolestaan luvussa 3.5 esiteltävässä vastavirta-algoritmissa.

3.5 Vastavirta-algoritmi

Neuroverkon opettamisessa painojen säätely on yksi olennaisimmista osista oppimisprosessia. Luvussa 3.3 käsiteltiin operaatio, jolla painoja voidaan säätää hintafunktiosta laskettavien gradienttien avulla. Tähän mennessä ei kuitenkaan vielä ole

käynyt selväksi, miten nämä gradientit määritetään.

Hintafunktion gradientin tietyillä painojen arvoilla laskee vastavirta-algoritmi. Yksi täysi kierros neuroverkon opettamisessa koostuu neuroverkossa etenemisestä (forward propagation), tuloksen määrittämisestä, ja tämän tuloksen avulla taaksepäin siirtymisestä (back-propagation). Tuloksen perusteella voidaan laskea miten hyvin verkko onnistui tehtävässään käyttämällä hintafunktiota. Tämän jälkeen aletaan siirtyä taaksepäin verkossa, määrittäen hintafunktion osittaisderivaattaa yksittäisten painojen suhteen. Tässä laskennassa hyödynnetään differentiaalilaskennan ketjusääntöä [3, s. 206–207].

Vastavirta-algoritmin voidaan ajatella kuvaavan tapaa saada verkko vastaamaan yhtä opetusdatan tapausta. Esimerkiksi kissaa esittävän kuvan halutaan saavan luokaksi kissan. Tähän tulokseen päädytään, kun verkon painoja painotetaan niin, että kuvassa kissaa muistuttavat piirteet vaikuttavat lopullisissa tuloksissa kissaluokan todennäköisyyteen. Jos tätä prosessia toistettaisiin vain kissan kuville, lopulta verkko osaisi vain tunnistaa kissan kissaksi, mutta olisi hyödytön minkä tahansa muun luokitteluongelman edessä. Tästä johtuen vastavirta-algoritmia toistetaan useille eri tapauksille, jolloin verkko oppii yleistämään useille eri luokille yhtäaikaisesti. Painojen säätäminen tapahtuu siis keskiarvoistetusti kaikille opetusdatan tapauksille, ainakin teoriassa. Käytännössä on hyödyllisempää tehdä painojen muutoksia pienelle osajoukolle opetusdatasta kerrallaan, koska vastavirta-algoritmi on suhteellisen raskas ja aikaa vievä operaatio. Tulokset eivät ole yhtä tarkkoja kuin kaikelle datalle tehtäessä, mutta hyöty operaation nopeudessa tekee siitä silti kannattavan [3, s. 151–152]. Vastavirta-algoritmin lopputuloksena saadaan gradientit, joita voidaan käyttää painojen säätämiseen gradient descent -menetelmän avulla.

4. NEUROVERKKOJEN OPTIMOINTI

Tässä luvussa käsitellään neuroverkkojen optimoinnin teoriaa, ja esitellään joitakin yksittäisiä menetelmiä, joilla neuroverkkojen suorituskykyä voi parantaa.

4.1 Optimointi yleisesti

Paras strategia neuroverkon optimoinnin edistämiseksi ei aina ole optimointialgoritmin kehittäminen. Sen sijaan optimointia voi parantaa suunnittelemalla käytetyn mallin alusta alkaen sellaiseksi, että optimointi on helppoa. Käytännön elämässä on tärkeämpää valita helposti optimoitavissa oleva malli sen sijaan, että käytettäisiin tehokasta optimointialgoritmia. Suurin osa neuroverkkojen editysaskelista viimeisen 30 vuoden aikana on tapahtunut muuttamalla verkon mallia, eikä suinkaan optimointimenetelmiä vaihtamalla. Stokastinen kaltevuuslasku (stochastic gradient descent) on pysynyt käytössä 1980-luvulta asti. [3]

Modernit neuroverkot käyttävät lineaarisia muunnoksia verkon kerrosten välissä, sekä aktivointifunktioita jotka ovat differentioitavissa lähes missä tahansa. Viimeaikaiset innovaatiot ovat saaneet lineaaristen funktioiden käytön nousemaan ja vastaavasti sigmoidisten funktioiden käytön laskuun. Nämä mallit omaavat hyviä ominaisuuksia, jotka helpottavat verkon optimointia. Toisin sanoen, modernit neuroverkot on suunniteltu niin, että lokaali gradientti-informaatio vastaa suhteellisen hyvin kaukaista ratkaisua kohti liikkumista. [3]

Monet ongelmat neuroverkkojen optimoinnissa nousevat hintafunktion globaalista luonteesta, eivätkä nämä ongelmat ole korjattavissa parantamalla lokaaleja suuntaarvioita. Yleisesti käytössä oleva tapa selättää tämä ongelma on parametrien alustaminen alueella joka on yhteydessä ratkaisuun lyhyellä polulla sellaisen parametriavaruuden läpi, joka on lokaalin gradienttilaskun tiedossa. [3]

4.2 Katoavan ja räjähtävän gradientin ongelma

Etenkin syvien neuroverkkojen opetuksessa verkon hintafunktion eri gradientteja selvitettäessä lopputulokseen päädytään useiden perättäisten laskutoimitusten seu-

rauksena. Toistuva laskutoimitusten sarja samoilla parametreilla aiheuttaa selkeitä ongelmia. Vastavirta-algoritmin osana jotakin arvoa saatetaan käyttää kertolaskun osana useita kertoja peräkkäin. Jos jokin kerroin W kerrotaan itsellään t kertaa, vastaa se sitä, että jotakin lukua kerrottaisiin luvulla W^t. Mikäli luku W poikkeaa merkittävästi yhdestä, saadaan lopputuloksena joko hyvin suuri tai hyvin pieni luku. Osana gradienttia tällainen luku aiheuttaa joko gradientin katoamisen tai gradientin räjähtämisen. Katoavan gradientin tapauksessa oppiminen hidastuu tai pysähtyy täysin, sillä gradientti on lähes vaakatasossa. Räjähtävä gradientti puolestaan saa aikaiseksi täysin holtittomia gradient descent -askelia, sillä se on lähes pystysuora. [3, s. 289–290, 402]

Syvien neuroverkkojen yleistyessä katoavan ja räjähtävän gradientin ongelmalle on kehitetty ratkaisuja. Yksi niistä, eränormalisointi, on esitelty tarkemmin luvussa 4.4.

4.3 Esiopetus

Joskus erityisen hankalan ongelman ratkaisemiseksi ei kannata opettaa yksittäistä neuroverkkoa. Sen sijaan voi olla hyödyllistä jakaa opetusprosessi eri abstraktiotasoille, eli käyttää jonkinlaista esiopetettua verkkoa [3, s. 322].

Yksi tapa käyttää esiopetettua verkkoa on määrittää ne asiat, joita datasta halutaan saada eroteltua, ja tehdä verkko joka oppii nuo asiat. Esimerkiksi kuvista opittaessa tämä voisi tarkoittaa, että kuvista halutaan ensin irti luokiteltava hahmo ja niin vähän taustaa kuin mahdollista. Tällaisen arkkitehtuurin kasaaminen itse on suhteellisen työlästä, sillä esiopetusverkon kasaaminen täytyy aloittaa tyhjästä.

Toinen tapa käyttää esiopetusta hyödyksi uuden ongelman ratkaisemisessa on hyödyntää jotakin valmiiksi opetettua verkkoa. Esimerkiksi Keras-kirjasto tarjoaa useita eri neuroverkkoja, jotka on jo valmiiksi opetettu ImageNet-aineistolla [7]. Tällaiselta verkolta voi kysyä suoraan luokittelun tuloksia, mutta mikäli käsillä olevan luokitteluongelman aineisto eroaa ImageNet-aineistosta, tämä ei vielä tuota haluttuja tuloksia.

Eri aineistoa käytettäessä on kuitenkin silti mahdollista hyödyntää ImageNet-aineistolla opetettua verkkoa, sillä valmista verkkoa pystyy käyttämään myös piirteiden irroituksessa. Koska ImageNet-aineiston pohjalta opetettu neuroverkko pyrkii luokittelemaan kuvia, sitä pystyy hyödyntämään monenlaisessa kuvanluokitteluongelmassa. Vaikka ongelman luokat eroaisivat ImageNet-luokista, onnistuneesti valitut piirteet ovat jotakuinkin samanlaisia oli sitten kyse mistä tahansa luokittelusta. Valmista

verkkoa käytettäessä päästään usein hyviin tuloksiin, ilman että esiopetukseen käytettävää verkkoa tarvitsee suunnitella alusta asti itse.

Esiopetuksen on todettu parantavan neuroverkon kykyä yleistää testidatan perusteella globaaliin tapaukseen. Esiopetettu neuroverkko oppii huomattavan erilaiset piirteet kuin verkko, jossa esiopetusta ei ole käytetty. [8]

4.4 Eränormalisointi

Eränormalisointi (batch normalization) on eräs uudempi menetelmä neuroverkkojen saralla. Sen tavoitteena on painottaa neuroverkon kerroksien tuottamia arvoja niin, että niiden jakauma pysyy samankaltaisena jokaisessa kerroksessa [9]. Tämän seurauksena on vähemmän todennäköistä, että optimointimenetelmä, kuten esimerkiksi luvussa 3.3 esitelty stochastic gradient descent, päätyy tasaiselle alueelle ja oppiminen hidastuu tai pysähtyy miltei täysin.

Eränormalisoinnin ansiosta verkon opettamiseen voidaan käyttää suurempaa oppimisnopeutta (learning rate) [9, s. 5], ja malli oppii yleistämään paremmin [10]. Käytettäessä suurta oppimisnopeutta, ongelmaksi nousee usein liian suuriksi kasvavat gradientin arvot. Eränormalisointia käytettäessä verkon parametreilla ei ole vaikutusta vastavirta-algoritmin tuottamiin tuloksiin [9, s. 5].

4.5 Hyperparametrit

Niitä neuroverkon parametreja, joiden säätämisestä ei ole vastuussa neuroverkko itse, kutsutaan hyperparametreiksi [3, s. 120]. Tällaisia parametreja ovat esimerkiksi verkon kapasiteetti, optimointimenetelmän oppimisnopeus (learning rate), painojen heikkeneminen (weight decay) ja yksittäisten kerroksien parametrit, kuten neuronien määrä ja konvoluutioikkunan koko.

Hyperparametrien säätäminen jää usein ihmisen vastuulle, ja optimointi on yleensä ad hoc -tyyppistä johtuen siitä, että moni koneoppimisongelma on niin tuntematon, ettei ennakkotapausta ole vielä olemassa. Kukaan ei siis välttämättä ole vielä yrittänyt löytää optimaalista verkkoa kyseisen ongelman ratkaisemiseksi. Neuroverkon käyttäytyminen uuden ongelman kanssa on vaikeaa ennakoida, joten monesti ainoa vaihtoehto on manuaalisesti käydä läpi erilaisia vaihtoehtoja ja yrittää löytää jokin toimiva parametrijoukko.

5. KOKEELLINEN OSUUS

Kokeellisen osuuden tarkoitus on pyrkiä hyödyntämään erilaisia neuroverkon optimointimenetelmiä. Optimoinnin kohteeksi on valittu Kaggle-alustan Dog Breed Identification -kilpailu, jossa kilpailun tavoite on pystyä tunnistamaan eri koirarotuja nimeltä mahdollisimman tarkasti. Kaggle-alustalta on ladattavissa opetusdata, jonka avulla mahdollinen oppimismenetelmä voidaan opettaa. Opetusdata sisältää hieman yli 10000 kuvaa koirista, ja kuvissa esiintyy 120 eri koirarotua. [11]

5.1 Työkalujen esittely

Työssä käytetään Python-ohjelmointikieltä ja Keras-kirjastoa. Pythonin käyttö tieteellisessä laskennassa on viime vuosina yleistynyt huomattavasti, ja se onkin tällä hetkellä yksi käytetyimmistä ohjelmointikielistä datatieteiden alalla [12]. Pythonin ohella käytetään myös sellaisia Python-kirjastoja kuin Scikit-learn ja Numpy. Molemmat näistä kirjastoista on saatavilla Pip-paketinhallintajärjestelmän kautta.

Keras on korkean tason neuroverkkoihin keskittynyt ohjelmointirajapinta [13]. Keras-rajapinta käyttää hyväkseen jotakin matalan tason laskentakirjastoa, kuten Theano-, CNTK- tai TensorFlow-kirjastoa. Tässä kokeessa käytössä on TensorFlow. Matalan tason laskentakirjastoissa on joitakin yksilöllisiä eroja. Esimerkiksi tässä työssä käytetty Xception-neuroverkko on saatavilla vain TensorFlow-alustalla, sillä se tarvitsee TensorFlow-kirjaston SeparableConvolution-kerrosta toimiakseen [7].

5.2 Kerasin Sequential-malli

Keras-kirjasto on kirjoitettu Python-ohjelmointikielellä, ja siten sen rajapintakutsuja on myöskin mielekästä tehdä Pythonilla. Keras tarjoaa yksinkertaisen rajapinnan neuroverkon rakentamista varten. Kerasin Sequential-malli mahdollistaa useamman kerroksen linkittämisen toisiinsa. Neuroverkkojen tapauksessa kyse on yleensä useamman konvoluutiokerroksen ja loppuun muutaman tiheän kerroksen linkittämisestä.

```
# Alusta malli.
model = Sequential()
# Lisää malliin yksi kaksiulotteinen konvoluutiokerros, joka ottaa sisäänsä
# 256x256 RGB-kuvia.
model.add(Conv2D(filters = 32,
                 activation = 'relu',
                 input_shape = (256, 256, 3),
                 kernel_size = (3,3)
# Lisää malliin lopullinen kerros, joka tuottaa kymmenen elementtiä
# pitkän vektorin. Tässä tapauksessa luokkia olisi siis kymmenen,
# ja tulosvektori sisältäisi todennäköisyyden jokaiselle luokalle.
model.add(Dense(10, activation = 'sigmoid'))
```

Lähdekoodi 5.1 Esimerkki Sequential-mallin alustuksesta

Kerasissa Sequential-malli ja neuroverkon kerrokset alustetaan lähdekoodin 5.1 esittämällä tavalla. Malliin voidaan lisätä erilaisia kerroksia, kuten kaksiulotteisia konvoluutiokerroksia sekä tiheitä kerroksia.

5.3 Mallin opettaminen

Opetusdata sisältää noin kymmenen tuhatta kuvaa joissa esiintyy 120 eri koirarotua. Datasta saadaan noin 83 kuvaa yksittäistä rotua kohden. Tällainen kuvamäärä yhtä luokkaa kohden on suhteellisen pieni. Koneoppimisen alalla kuuluisan ImageNet-haasteen luokitteluongelman opetusdataan kuuluu vuonna 2017 1.2 miljoonaa opetuskuvaa joissa esiintyy tuhat eri luokkaa [14]. Tämä tarkoittaa, että jokaista luokkaa kohden on tarjolla 1200 eri kuvaa.

Pienestä datamäärästä on vaikeaa oppia eri rotujen välisiä eroja pelkästään hyvän mallin kasaamalla. Jo kokeen lähestymistapaa suunniteltaessa oli selvää, että jotain valmista dataa kannattaa käyttää hyödyksi. Keras-kirjasto tarjoaa helppokäyttöisen rajapinnan esiopetettuihin malleihin. Työssä päädyttiin käyttämään VGG19-, InceptionResNetV2- ja Xception-mallia datan esikäsittelyyn. Kerasin esiopetettujen verkkojen rajapintakutsuihin kuuluu datan esikäsittelyä varten tarkoitettu preprocess_input-funktio, joka ottaa parametrina 4D-Numpy-taulukon, joka sisältää RGB-kuvia. RGB-kuva esitetään kolmena matriisina, jotka sisältävät kokonaislukuarvoja väliltä 0 ja 255. Kerasin esiopetetun neuroverkon rajapintakutsut on esitelty koodissa 5.2.

```
def extract_features_Xception(x_train):
    # Muokkaa opetusdata Xception-mallin ymmärtämään muotoon.
    preprocessed_input = xception.preprocess_input(x_train)
    # Lataa ImageNet-datalla esiopetettu Xception-neuroverkko.
    model = Xception(weights='imagenet', include_top=False)
    # Palauta Xception-verkon tuottamat piirteet.
    return model.predict(preprocessed_input, batch_size=32)
```

Lähdekoodi 5.2 Esimerkki piirteiden irroituksesta esiopetetulla verkolla.

Preprocess_input-funktio palauttaa mallin käsittelyn seurauksena kuvista saadut piirteet (features). Jokaisen esiopetetun verkon palauttamat piirteet menevät ensin läpi tiheästä kerroksesta. Tämän yhteydessä datalle suoritetaan Keras-kirjaston GlobalAveragePooling2D-operaatio ja eränormalisointi (Kerasissa BatchNormalization). Tämän jälkeen syötteet yhdistetään Kerasin Concatenate-funktiolla. Viimeisten kerroksien ajan data on siis liitettynä yhteen matriisiin, jokainen yksittäinen piirrejoukko omassa kerroksessaan. Tämä yhdistetty data menee läpi tiheästä kerroksesta ja sille tehdään vielä yksi eränormalisointi, jonka jälkeen päästään lopulliseen tulokset tuottavaan ulostulokerrokseen. Lopputuloksena saadaan kuvien luokitteluennusteet 120 elementtiä pitkässä vektorissa. Vektori sisältää siis mallin ennustaman todennäköisyyden jokaiselle koirarodulle.

Tätä lopputulosta voidaan vertailla oikeiden vastauksien kanssa niin, että suurin todennäköisyys tulkitaan mallin arvaukseksi. Vertailun tuloksena saadaan prosenttiosuus oikein arvatuista koiraroduista. Mallilta kyseltävät kuvat ovat sellaisia, joita sille ei opettamisen yhteydessä näytetty ollenkaan. Toisin sanoen tällainen testaaminen mittaa miten hyvin malli suoriutuu kuvista, joita se ei ole ennen nähnyt. On tosin huomionarvoista, että siltikin on mahdollista, että tulokset eivät täysin vastaa reaalimaailman tilannetta. Tämä johtuu siitä, että opetusdatan kuvat eivät aina välttämättä ole täysin sattumanvaraisia, tai omaavat joitakin piirteitä joita uusissa ennen näkemättömissä kuvissa ei ole. Esimerkiksi sellaiset asiat kuten kohteiden etäisyys, asento, kuvan rajaus ja valaistus voivat poiketa testidatan ja reaalimaailman välillä. Tämän kokeilun kannalta on kuitenkin mielekästä olettaa, että testidatan kuvat vastaavat hyvin myös testidatan ulkopuolisia, uusia kuvia.

5.4 Tulosten parantaminen

Kun mallin avulla voidaan tuottaa jokin tulos, on mahdollista alkaa parantamaan kyseistä tulosta. Pyrkimyksenä kokeilussa oli, että tulosten parantaminen aloitetaan

osa-alueesta, jolla on suurin vaikutus tuloksiin. Ensimmäiseksi kehityskohteeksi valikoitui siis verkon rakenne.

Verkon rakenne pyrittiin rakentamaan olemassa olevien verkkojen tyyliseksi. Erilaisten kuvanluokitteluongelmien ratkaisemiseksi on tehty paljon tutkimustyötä, joten tätä valmista tietoa kannattaa hyödyntää tämänkin ongelman ratkaisemisessa. Ennen kaikkea valmiiksi opetettujen verkkojen käyttäminen koettiin hyödylliseksi, sillä se tarjoaa mahdollisuuden hyödyntää suuria määriä kuvadataa, siinä missä koirarotuaineisto on suhteellisen suppea. Kokeessa käytetyt valmiit verkot olivat opetettu ImageNet-aineistolla, ja se sisältää jonkin verran kuvia myös koirista. Siinä mielessä on perusteltua ajatella verkon pystyvän irroittamaan sellaisia piirteitä, joiden avulla koiria pystyy tunnistamaan.

Käytännössä valmiiksi opetettujen verkkojen käytöllä oli valtava vaikutus mallin oppimiseen. Ensimmäiset vedokset mallista pystyivät vain vähän sattumanvaraista arvailua parempiin tuloksiin. Kun ensimmäinen esiopetettu verkko otettiin mukaan malliin, tulokset nousivat n. 60% osumatarkkuuteen. Oli siis selvää, että esiopetettujen verkkojen käyttäminen on hyvä idea tällaisessa tehtävässä.

Valmiissa verkossa käytetään paljon luvussa 4.4 käsiteltyä eränormalisointi (batch normalization) -menetelmää. Tämä mahdollisti osaltaan näinkin syvän verkon käyttämisen. Myös vielä syvempää rakennetta kokeiltiin, mutta vaikutus tuloksiin oli lähinnä negatiivinen.

Viimeinen kehityskohde verkon rakenteen saatua lopullisen muotonsa oli hyperparametrien säätäminen. Mallissa päädyttiin käyttämään Keras-rajapinnan RMSprop-optimointimenetelmää. Menetelmän oppimisnopeutta laskettiin oletusarvosta 0.001 arvoon 0.00002. Tällä oli selvä vaikutus mallin oppimiseen. Oletusarvolla malli ylioppi nopeasti jo ennen kymmentä iteraatiokierrosta (epoch). Oppimisnopeutta pienentämällä saatiin validointidatan virheeksi (loss) huomattavasti pienempi arvo kuin alkuperäisellä arvolla.

6. TULOKSET

Kokeen lopulliseksi onnistumisprosentiksi saatiin 79.1%. Toisin sanoen malli pystyisi tunnistamaan noin neljä viidestä opetusdatan joukossa olevasta koirarodusta oikein, jos sille esitettäisiin uusi testidatan ulkopuolinen kuva koirasta.

Toisaalta tulos on hyvä, sillä vastaava prosentti valtaosalle ihmisväestöä on todennäköisesti huonompi. Edes koirarotuihin paremmin perehtynyt henkilö tuskin pystyisi aina erottamaan tiettyjä hyvin samannäköisiä koirarotuja toisistaan. Lisäksi aineistoa oli melko rajattu määrä, ja neuroverkkojen oppiminen on hyvin oleellisesti riippuvainen opetukseen käytetyn datan määrästä.

Tulos voisi todennäköisesti olla myös huomattavasti parempi. Esimerkiksi kuvien esikäsittely erilaisilla neuroverkoilla voisi tuottaa hyviä tuloksia. Samoin opetusdatan esikäsittely ja keinotekoinen laajentaminen saattaisi parantaa mallin kykyä yleistää uusiin kuviin. Tähän datan monistamiseen ja manipuloimiseen on tarjolla valmis ratkaisu Keras-kirjastossa. Datan manipulointi olisi saattanut olla hyödyksi tässäkin tapauksessa.

Yksi mallin tehokkuutta haittaava tekijä saattoi olla se, että kuvia jouduttiin pienentämään käytännöllisistä syistä. Kokeilua tehtiin tietokoneella, jossa oli vain kahdeksan gigatavua keskusmuistia. Esimerkiksi 224x224-kokoiset kuvat aiheuttivat muistin loppumisen verkkoa opetettaessa, ja sen seurauksena Python-prosessin kaatumisen. Kuvien kokoa jouduttiin pienentämään kokoon 160x160, jotta testiaineisto saatiin ajettua läpi neuroverkosta. Vaihtoehtoisia ratkaisuja muistinloppumisongelmaan olisi voinut olla datan väliaikainen tallentaminen levylle, mutta tämä ei välttämättä ole mahdollista jos muistin loppuminen tapahtuu neuroverkon opetusvaiheessa. Keras saattaa vaatia kaiken datan pysymisen muistissa, sillä dataa tarvitaan jatkuvasti neuroverkkoa opetettaessa. Lisäksi levyltä lukeminen olisi huomattavasti hitaampaa kuin keskusmuistista.

Kaiken kaikkiaan kokeilu osoitti, että nykyaikaiset neuroverkot pystyvät oppimaan tällaisen luokitteluongelman suhteellisen pienellä vaivalla. Kandidaatintyön laajudessa ei varmastikaan pystytä käsittelemään kaikkia niitä vaihtoehtoja ja eri toimin-

tamalleja, joilla kokeilun tuloksia saisi vielä huomattavasti paremmaksi. Jo näinkin pintapuolisella kokeilulla huomattiin kuitenkin, että neuroverkko pystyy onnistumaan ihailtavalla tarkkuudella näin hankalasta tehtävästä.

LÄHTEET

[1] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *CoRR*, vol. abs/1202.2745, 2012. [Online]. Available: http://arxiv.org/abs/1202.2745

[2] T. M. Mitchell, "The discipline of machine learning," Machine Learning Department, Carnegie Mellon University, Tech. Rep., 2006. [Online]. Available: http://www.cs.cmu.edu/~tom/pubs/MachineLearning.pdf

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] P. L. Lanzi, "Fast feature selection with genetic algorithms: A filter approach," 1997. [Online]. Available: https://pdfs.semanticscholar.org/cf63/0dbd510c5203bddd4a40c51c0e3ec47b6cfa.pdf

[5] "Vgg in tensorflow," accessed: 2017-12-17. [Online]. Available: http://www.cs.toronto.edu/~frossard/post/vgg16/

[6] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in neural information processing systems*. Morgan Kaufmann, 1992, pp. 950–957. [Online]. Available: https://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.pdf

[7] "Keras applications," accessed: 2017-12-17. [Online]. Available: https://keras.io/applications

[8] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, and P. Vincent, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, 2010. [Online]. Available: http://www.jmlr.org/papers/volume11/erhan10a/erhan10a.pdf

[9] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[10] M. Simon, E. Rodner, and J. Denzler, "Imagenet pre-trained models with batch normalization," *CoRR*, vol. abs/1612.01452, 2016. [Online]. Available: http://arxiv.org/abs/1612.01452

[11] "Dog breed identification," accessed: 2017-12-17. [Online]. Available: https://www.kaggle.com/c/dog-breed-identification

[12] "Data science/machine learning tool usage poll," accessed: 2017-12-17. [Online]. Available: https://www.kdnuggets.com/2017/05/poll-analytics-data-science-machine-learning-software-leaders.html

[13] "Keras documentation," accessed: 2017-12-17. [Online]. Available: https://keras.io/

[14] "Large scale visual recognition challenge 2017," accessed: 2017-12-17. [Online]. Available: http://image-net.org/challenges/LSVRC/2017/index